down a runway; likewise, consider a new financial analyst learning-by-doing as he structures a multi-million dollar financial loan. Few employers are willing to endure such failures to have a more competent workforce.

The key to such a support system is that it is seamlessly integrated into the business system that the knowledge worker uses to execute their job tasks. Workers don't need to go "off-line" or seek out cryptic information buried within paper manuals and binders for guidance or to find the answer to queries. All the support components are made available through the same applications the worker's use, at the point in which they need them, tailored to the individual to show "how", not just "what". Learning would be occurring all the time, with little distinction between performing and improving performance. Establishing that training should focus on performance (how), rather than facts (what), and extending the model of learning to include assistance while performing, rather than only before performance, still leaves us dangerously exposed in preparing to compete in the new, chaotic economy. As was mentioned in the opening of this paper, the pace of change in business today is whiplash fast. Not only are new methods of doing business evolving every 18-24 months, new competitors emerge, dominate, and fade in time periods businesses used to take to perform demographic studies. Now more than ever, those who do not reinvent themselves on a regular basis will be fossilized by the pace of change. A typical BusSim engagement takes between one and two years to complete and requires a variety of both functional and technical skills. Figure 3 depicts the timeline and relative resource requirements for each phase of development for a typical application development in accordance with a preferred embodiment. The chart clearly depicts the relationship between the large number of technical resources required for both the build and test phases of development. This is because the traditional development process used to build BusSim solutions reflects more of a "one off" philosophy, where development is done from scratch in a monolithic fashion, with little or no reuse from one application to the next. This lack of reuse makes this approach prohibitively expensive, as well as lengthy, for future BusSim projects.

The solution to this problem is to put tools in the hands of instructional designers that allows them to create their BusSim designs and implement them without the need for programmers to write code. And to put application architectures that integrate with the tools in the hands of developers, providing them with the ability to quickly deliver solutions for a number of different platforms. The reuse, then, comes in using the tools and architectures from one engagement to another. Both functional and technical resources carry with them the knowledge of how to use the technology, which also has an associated benefit of establishing a best-practice development methodology for BusSim engagements.

Development Cycle Activities

In the Design Phase, instructional designers become oriented to the content area and begin to conceptualize an instructional approach. They familiarize themselves with the subject matter through reading materials and interviews with Subject Matter Experts (SMEs). They also identify learning objectives from key client contacts. Conceptual designs for student interactions and interface layouts also begin to emerge. After the conceptual designs have taken shape, Low-Fi user testing (a.k.a. Conference Room Piloting) is performed. Students interact with interface mock-ups while facilitators observe and record any issues. Finally, detailed designs are created that incorporate findings. These detailed designs are handed off to the development team for implementation. The design phase has traditionally been fraught with several problems. Unlike a traditional business system, BusSim solutions are not rooted in tangible business processes, so requirements are difficult to identify in a concrete way. This leaves instructional designers with a 'blue sky' design problem. With few business-driven constraints on the solution, shallow expertise in the content area, and limited technical skills, instructional designers have little help in beginning a design. Typically, only experienced designers have been able to conjure interface, analysis, and feedback designs that meet the learning objectives yet remain technically feasible to implement. To compound the problem, BusSim solutions are very open

ended in nature. The designer must anticipate a huge combination of student behavior to design feedback that is helpful and realistic.

During the build phase, the application development team uses the detailed designs to code the application. Coding tasks include the interfaces and widgets that the student interacts with. The interfaces can be made up of buttons, grids, check boxes, or any other screen controls that allow the student to view and manipulate his deliverables. The developer must also code logic that analyzes the student's work and provides feedback interactions. These interactions may take the form of text and/or multimedia feedback from simulated team members, conversations with simulated team members, or direct manipulations of the student's work by simulated team members. In parallel with these coding efforts, graphics, videos, and audio are being created for use in the application. Managing the development of these assets have their own complications. Risks in the build phase include misinterpretation of the designs. If the developer does not accurately understand the designer's intentions, the application will not function as desired. Also, coding these applications requires very skilled developers because the logic that analyzes the student's work and composes feedback is very complex.

The Test Phase, as the name implies, is for testing the application. Testing is performed to verify the application in three ways: first that the application functions properly (functional testing), second that the students understand the interface and can navigate effectively (usability testing), and third that the learning objectives are met (cognition testing). Functional testing of the application can be carried out by the development team or by a dedicated test team. If the application fails to function properly, it is debugged, fixed, recompiled and retested until its operation is satisfactory. Usability and cognition testing can only be carried out by test students who are unfamiliar with the application. If usability is unsatisfactory, parts of the interface and or feedback logic may need to be redesigned, recoded, and retested. If the learning objectives are not met, large parts of the application may need to be removed and completely redeveloped from a different perspective. The test phase is typically where most of the difficulties in the BusSim development cycle are encountered. The process of discovering and fixing functional, usability, and cognition problems is a difficult process and not an exact science.

For functional testing, testers operate the application, either by following a test script or by acting spontaneously and documenting their actions as they go. When a problem or unexpected result is encountered, it too is documented. The application developer responsible for that part of the application then receives the documentation and attempts to duplicate the problem by repeating the tester's actions. When the problem is duplicated, the developer investigates further to find the cause and implement a fix. The developer once again repeats the tester's actions to verify that the fix solved the problem. Finally, all other test scripts must be rerun to verify that the fix did not have unintended consequences elsewhere in the application.The Execution Phase refers to the steady state operation of the completed application in its production environment. For some clients, this involves phone support for students. Clients may also want the ability to track students' progress and control their progression through the course. Lastly, clients may want the ability to track issues so they may be considered for inclusion in course maintenance releases

One of the key values of on-line courses is that they can be taken at a time, location, and pace that is convenient for the individual student. However, because students are not centrally located, support is not always readily available. For this reason it is often desirable to have phone support for students. Clients may also desire to track students' progress, or control their advancement through the course. Under this strategy, after a student completes a section of the course, he will transfer his progress data to a processing center either electronically or by physically mailing a disk. There it can be analyzed to verify that he completed all required work satisfactorily. One difficulty commonly associated with student tracking is isolating the student

data for analysis. It can be unwieldy to transmit all the course data, so it is often imperative to isolate the minimum data required to perform the necessary analysis of the student's progress.

A Delivery Framework for Business Simulation

As discussed earlier, the traditional development process used to build BusSim solutions reflects more of a "one off" philosophy, where development is done from scratch in a monolithic fashion, with little or no reuse from one application to the next. A better approach would be to focus on reducing the total effort required for development through reuse, which, in turn would decrease cost and development time. The first step in considering reuse as an option is the identification of common aspects of the different BusSim applications that can be generalized to be useful in future applications. In examination of the elements that make up these applications, three common aspects emerge as integral parts of each: Interface, Analysis and Interpretation. Every BusSim application must have a mechanism for interaction with the student. The degree of complexity of each interface may vary, from the high interactivity of a high-fidelity real-time simulation task, to the less complex information delivery requirements of a business case background information task. Regardless of how sophisticated the User Interface (UI), it is a vital piece of making the underlying simulation and feedback logic useful to the end user.

Every BusSim application does analysis on the data that defines the current state of the simulation many times throughout the execution of the application. This analysis is done either to determine what is happening in the simulation, or to perform additional calculations on the data which are then fed back into the simulation. For example, the analysis may be the recognition of any actions the student has taken on artifacts within the simulated environment (notebooks, number values, interviews conducted, etc.), or it may be the calculation of an ROI based on numbers the student has supplied. Substantive, useful feedback is a critical piece of any BusSim application. It is the main mechanism to communicate if actions taken by the student are helping or hurting them meet their performance objectives. The interpretation piece of the set of proposed commonalties takes the results of any analysis performed and makes sense of it. It takes the non-biased view of the world that the Analysis portion delivers (i.e., "Demand is up 3%") and places some evaluative context around it (i.e., "Demand is below the expected 7%; you're in trouble!", or "Demand has exceeded projections of 1.5%; Great job!").

There are several approaches to capturing commonalties for reuse. Two of the more common approaches are framework-based and component-based. To help illustrate the differences between the two approaches, we will draw an analogy between building an application and building a house. One can construct a house from scratch, using the raw materials, 2x4s, nails, paint, concrete, etc. One can also construct an application from scratch, using the raw materials of new designs and new code. The effort involved in both undertakings can be reduced through framework-based and/or component-based reuse. Within the paradigm of framework-based reuse, a generic framework or architecture is constructed that contains commonalties. In the house analogy, one could purchase a prefabricated house framework consisting of floors, outside walls, bearing walls and a roof. The house can be customized by adding partition walls, wall-paper, woodwork, carpeting etc. Similarly, prefabricated application frameworks are available that contain baseline application structure and functionality. Individual applications are completed by adding specific functionality and customizing the look-and-feel. An example of a commonly used application framework is Microsoft Foundation Classes. It is a framework for developing Windows applications using C++. MFC supplies the base functionality of a windowing application and the developer completes the application by adding functionality within the framework. Framework-based reuse is best suited for capturing *template-like* features, for example user interface management, procedural object behaviors, and any other features that may require specialization. Some benefits of using a framework include:

Extensive functionality can be incorporated into a framework. In the house analogy, if I know I am going to build a whole neighborhood of three bedroom ranches, I can build the plumbing, wiring, and partition walls right into the framework,